not a match (and succeeding for any device for which there is a match). In this way the filter driver "masks" LUNs, i.e., prevents the host from accessing unassigned LUNs.

Another service provided by the SAN manager of the invention relates to File System monitoring and extension. With reference to FIGURE 11, A SAN Storage Automation Service module 78 (SANStorAuto) functions as a controller for policy information. In that capacity, it has three main functions, namely, (1) maintenance of policies, (2) notification to File System monitor module 80 (FSMonitor) of policy changes, and (3) processing events when policies are exceeded.

The SANStorAuto 78 maintains a set of database tables that indicate the current policy definitions for each managed host. This policy includes a monitor flag, extend flag, maximum file system size, threshold, alert interval, LUN type, lower bound and upper bound.

A SAN Administrator Client module 82 (SANAdminClient) can request policy information from SANStorAuto 78 to be displayed on a graphical user interface console (not shown) and can send policy updates back to be saved in a database. When policy updates are made via the GUI, they are pushed down to the corresponding file system monitors.

When a file system monitor detects that a policy has been exceeded, an event is sent to the SANStorAuto 78. The policy engine 38a receives this event and determines if the file system can and/or should be extended, or if only notification is required. If the file system should be extended, then the policy engine determines what LUN to use and requests that the LUN be assigned to by the SANLunMgr 72. Once the LUN is assigned, a File System Extension service (SANAgenFSExtend) 84 is called to perform the extension by utilizing the host local operating system to extend the file system onto the newly assigned LUN.

A SANAgentScheduler 86 is a utility function that lets other functions schedule actions to be started some time in the future. It maintains a list of activity requests and the action to be performed when the request time is reached.

At startup, a SANDBParms utility service 88 retrieves database parameters from the TMD and stores them as an object. Other services can then access the object to create database connections. There is also a helper functions for creating a pool of database connections that can be reused.

A SANIndex 90 is a utility service that maintains a database table that other services can create, named sequences in. It will return the next index value given a sequence name.

A SANEvent is a utility service that can perform 3 functions, namely, (1) logs all SANEvents, (2) forwards events to SNMP and TEC, and (3) maintains the location of the SNMP and TEC event consoles.

SANEvent service subscribes to all SANEvents. All other events published by TSNM extend SANEvent. When a SANEvent is received, it is logged in the TKS message log.

SANEvent service will look inside each SANEvent it receives and if there is SNMP and or TEC information in the SANEvent, the events will be forwarded to the SNMP or TEC consoles.

Another function of SANEventService is to maintain the location of the SNMP and TEC consoles. The SANCommonAdminClient requests the location information to be displayed on the Console and sends updates back.

Peer Classes and Component Data Persistence

The SAN manager of the invention preferably utilizes an Object Oriented (OO) data model, but employs a relational data model for storing persistent data. The SAN manager employs peer classes, as discussed in more detail below, to map the OO model onto a relational model. The use of peer classes advantageously isolates the business logic from the relational database logic while allowing the use of inheritance in the business and database logic. This has the added advantage that different third party products for mapping an OO model to a relational model can be utilized without impacting the business logic.

With reference to FIGURES 12 and 13, the use of peer classes in accord with the teachings of the invention for mapping an OO model to a relational model can be better understood by considering an example. FIGURE 12 illustrates a simple object model 90 including an inheritance tree with two abstract classes 92 and 94 and a concrete class 96. Each class 92 -- 96 includes persistable data (a1, a2, and a3).